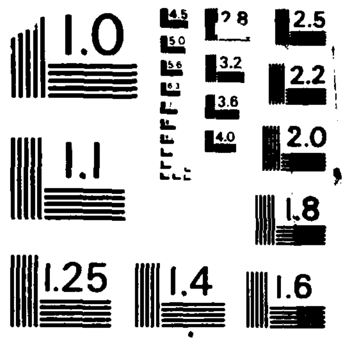
AD-A179 537 PROBABILISTIC ANALYSIS OF ALGORITHMS FOR NP-COMPLETE PROBLEMS(U) INDIANA UNIV AT BLOOMINGTON DEPT OF COMPUTER SCIENCE J FRANCO OCT 86 AFOSR-TR-87-0406 1/1

UNCLASSIFIED AFOSR-84-0372 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

DTIC FILE COPY

2

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

AD-A179 537

DTIC SELECTED APR 2 4 1987 D

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 87-0406
6a. NAME OF PERFORMING ORGANIZATION Indiana University	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION AFOSR/NM	
6c. ADDRESS (City, State and ZIP Code) Department of Computer Science Bloomington, IN 47405	7b. ADDRESS (City, State and ZIP Code) Bldg 410 Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-84-0372	
8c. ADDRESS (City, State and ZIP Code) Bldg 410 Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A3	

11. TITLE (Include Security Classification) Probabilistic Analysis of Algorithms for NP-complete Problems

12. PERSONAL AUTHOR(S) Professor J. Franco

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Annual	FROM 10/85 TO 9/86	October 86	2

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The probabilistic performance of a number of algorithms for the NP-complete satisfiability problem (SAT) has been investigated analytically and experimentally using a fixed-clause-length model generating n clauses of k - 3 literals taken from r variables as well as a random-clause-length model generating n clauses containing each of r variables independently with probability p. In the case of the random-clause-length model one polynomial time algori has been shown to find a solution to a random instance I of SAT with probability approaching 1 as in and r get large when a solution exists for I. In the case of the fixed-clause-length model, we have discovered an algorithm which almost always finds a solution to random satisfiable instances of SAT with k = 3. We have also shown that none of a wide class of algorithms can verify unsatisfiability in polynomial time almost always.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Captain Thomas	(202) 767-5026	NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AFOSR-TR- 87-0406

UNITED STATES AIR FORCE
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
BUILDING 410, BOLLING AFB, D.C. 20332

Grant No. AFOSR 84-0372

ANNUAL SCIENTIFIC REPORT

October 1985 to September 1986

Probabilistic Analysis of Algorithms for NP-complete Problems

John Franco, Principal Investigator
Department of Computer Science
Indiana University
Bloomington, Indiana 47405

87 4

TABLE OF CONTENTS

Abstract

The probabilistic performance of a number of algorithms for the NP-complete Satisfiability Problem (SAT) has been investigated analytically and experimentally using a fixed-clause-length model generating n clauses of $k > 3$ literals taken from r variables as well as a random-clause-length model generating n clauses containing each of r variables independently with probability p. In the case of the random-clause-length model one polynomial time algorithm has been shown to find a solution to a random instance I of SAT with probability approaching 1 as n and r get large when a solution exists for I. In the case of the fixed-clause-length model. we have discovered an algorithm which almost always finds a solution to random, satisfiable instances of SAT with $k = 3$. We have also shown that none of a wide class of algorithms can verify unsatisfiability in polynomial time almost always.

We have also studied the algorithm of Angluin and Valiant for the NP-complete Hamiltonian Circuit problem. It was shown by them that a hamiltonian circuit can be found in random graphs which are hamiltonian with probability tending to 1 as graph size tends to infinity. We have found that their algorithm almost never finds hamiltonian circuits in k-regular graphs which are hamiltonian. On the other hand, we have discovered an algorithm which often finds hamiltonian circuits in such graphs.

1. Research Objective

The goal of this research is to develop and analyze algorithms which can, in some practical sense, solve NP-complete problems quickly. NP-complete problems appear in many disciplines such as Cryptology, Operations Research, Artificial Intelligence and Computer System Design. NP-complete problems are the "hardest" of a class of problems known as NP. Associated with each NP problem we consider is an infinite set of instances. Instances may take the form of graphs, logic expressions, sets or many other structures depending on the problem. Each instance has a *size* denoted by n. Although the size of an instance I may be formally defined as the number of bits needed to efficiently encode I, for our purposes, we may regard the size of I to be the number of distinct objects in I. So, for example, a graph containing E edges and Q vertices has size $n = E + Q$. Associated with each instance I is a set of variables, a set of values that can be assigned to each variable and a constraint function U_I that maps value assignments to variables to $\{true, false\}$. For example, if I is a graph with Q vertices we might associate $Q - 1$ variables which take edge labels as values and a constraint function which has value *true* if and only if the edge set corresponding to the assignment given to the variables is a spanning tree of I. An assignment t such that $U_I(t) = true$ is a solution to I. An algorithm solves I if it determines whether or not a solution exists for I.

A problem in NP is said to be solved efficiently if there is an algorithm which solves every instance of the problem in time bounded by a polynomial in n. Unfortunately, there is no known computational scheme for efficiently solving any NP-complete problem and it is considered highly unlikely that one will be found (see [2] and [12]). Thus, every known method for solving an NP-complete problem P cannot find the solution to some instances of P in a reasonable amount of time. Furthermore, there is little hope that even an effective randomized algorithm (see [13], [22] and [23]) will be found for any NP-complete problem since, as is well known, this would imply an unlikely collapse of the polynomial hierarchy. However, if a method M can be found to efficiently find solutions to all but a few instances of P then M might be a practical method for solving P. We are looking for such (M, P) pairs.

2. Analytic Tools

We use probability theory to measure success in meeting our goal. A distribution D is assigned to the set of all possible instances of P of size n and we prove one of three kinds of results for a given algorithm M:

a. M finds a solution to an instance of P chosen randomly according to D in time bounded by a polynomial in n with probability greater than some positive constant κ as n gets large. Then we say M efficiently solves P *in bounded probability* under D.

b. M finds a solution to an instance of P chosen randomly according to D in time bounded by a polynomial in n with probability approaching 1 as n gets large (we will say "with probability tending to 1"). Then we say M efficiently solves P *in probability* under D.

c. M solves all of a large sample of instances of P chosen randomly according to D in average time that is bounded by a polynomial in n as n gets large. Then we say that M solves P *in polynomial average time.*

Results of type (a) are weaker than results of type (b) and results of type (b) are weaker than results of type (c). It is often the case that we can prove a weaker result but not a stronger one for a particular (M, P) pair under D. Although a type (c) result is the most desirable type of result, even a type (b) result will allow us to conclude that M, in some practical sense (at least under D), efficiently solves P. A result of type (a) cannot always allow us to make the same conclusion since κ may be very small (say .01). However, many algorithms we consider proceed by assigning values to variables in some order which is decided during computation and assignments are never undone either totally or partially. These algorithms either continue until all variables are assigned values (in which case a solution has been obtained) or they stop prematurely because they discover that every set of assignments of values to unassigned variabies cannot possibly lead to a solution (in which case it cannot be determined whether or not a solution exists). A property of these algorithms is that the next variable to be assigned a value is chosen randomly from a large group of possibilities. Thus, repeated runs of such algorithms will execute differently and possibly give different results. If the probability that a run finds a solution is bounded from below by a constant and all runs execute independently then only a constant number of runs would be necessary for us to solve a random instance of P with probability arbitrarily close to 1 (this can be strengthened to a type (b) result if the number of runs is allowed to grow slightly with n). Unfortunately, it is not the case that all runs execute independently. However, for the algorithms we consider, the dependence is very weak and, according to the results of our experiments, we are justified in supposing that a small number of repeated runs of M will allow us to solve P with probability tending to 1. Thus, a result of type (a) seems to translate to a result of type (b) for the kinds of algorithms we consider. When referring to results of either type (a), (b) or (c) we will sometimes use the phrase *"probabilistically efficient"*.

Others have taken this approach for specific NP-complete problems. Algorithms which are probabilistically efficient have been found for the Hamiltonian Circuit problem [1] and [19], the Planar Traveling Salesman problem [18], the Processor Scheduling problem [7], the Bin Packing problem [17] and other NP-complete problems.

3. Finding Solutions to Instances of SAT: Old and New Results

The problem we are primarily interested in is the Satisfiability problem (SAT). An instance I of SAT is a Boolean expression in Conjunctive Normal Form (CNF). A CNF expression is a conjunction (logical and) of disjunctions (logical or) of literals (a literal is a Boolean variable or its complement). A disjunction is also called a clause. A solution to I, if one exists, is a truth assignment to the variables associated with literals in I which cause I to have value true. The problem is to find a solution to I, if one exists, or to determine that no solution to I exists. An instance of SAT which has a solution is said to be ***satisfiable***; otherwise the instance is said to be ***unsatisfiable***. SAT is the first problem found to be NP-complete and is closely related to problems in Artificial Intelligence particularly in the areas of Theorem Proving and Vision Analysis. Also, any problem in NP can easily be transformed to SAT and transformations from SAT to other NP-complete problems are often straightforward. SAT is, therefore, one of the more important NP-complete problems.

Some favorable probabilistic results for algorithms which solve SAT have already been obtained. Let $V = \{v_1, v_2, \ldots, v_r\}$ be a set of r Boolean variables. A ***random clause*** contains each possible literal $v_1, \ldots, v_r, v'_1, \ldots v'_r$ with probability p independently of the occurrence of any other literal. A ***fixed length random clause*** of length k contains k distinct literals which are equally likely to be any k-subset of $2r$ literals associated with the variables of V such that no two literals are associated with the same variable.

The ***random clause model*** is the distribution on instances of SAT where each instance has n independently selected ***random clauses***. In [3], [14], [15], [20] and [21] the average running time of several algorithms for SAT is obtained under the random clause model. The conditions under which at least one algorithm runs in polynomial time on the average are as follows:

1) $\lim_{r\to\infty} rp = 0$, $n \geq r\ln(2)/-\ln((r+1)p)$.

2) $\lim_{r\to\infty} rp = \infty$, $\lim_{r\to\infty} p = 0$, $n \geq \ln(2)e^{2rp}/ep$.

3) $\lim_{r\to\infty} p = 0$, $np \leq \sqrt{\frac{c\ln(r)}{r}}$, c constant.

4) $\lim_{r\to\infty} 1/p =$ polynomial(r), $np \leq r^{cp}$, c constant.

5) $n \leq c\ln(r)$, c constant

But, in [9] we showed that, under the random clause model, a randomly chosen truth assignment to V nearly always is a solution to an instance of SAT when *a*) $p \geq \ln(n)/r$, and an instance of SAT nearly always has a clause containing no literals (such an instance is unsatisfiable) when *b*) $p \leq \ln(n)/(2r)$. We also showed that, when *c*) $\ln(n)/(2r) < p < \ln(n)/r$ and $n\ln(n) < \alpha\sqrt{r\ln(r)}$, where α is any constant greater than zero, a random instance of SAT has no variable which appears in more than one clause with probability tending to 1. Since a clause containing no variables is not satisfiable and since solutions to instances containing variables that appear in at most one clause are trivial to find, we may conclude that instances of SAT generated under either *a*), *b*) or *c*) may be trivially solved.

These results are significant because conditions *a*), *b*) and *c*) subsume conditions 1) – 5) above. Thus, our results indicate that the previous results on algorithms for SAT are favorable, not because the algorithms analyzed have some special property which make them fast in the probabilistic sense, but because the assumed distribution generates instances which have the property that almost any simple-minded algorithm can solve them efficiently almost all the time.

Let I be an instance of SAT and let c denote a clause of I. If v is a literal in I then we use $comp(v)$ to denote the complement of v. In order to express algorithms for SAT succinctly we regard clauses of I to be subsets of literals $\{v_1, ..., v_r, v'_1, ..., v'_r\}$ and I to be a collection of n of these subsets. In [16] we considered the following algorithm for SAT:

```
$A_1(I)$:
    While $I \neq \phi$ and $\forall c \in I,\ c \neq \phi$
        If there is a single-literal clause $\{u\} \in I$ then $v \leftarrow u$
        Else choose a literal $v$ randomly from $L$
        $I \leftarrow \{c - \{comp(v)\} : c \in I$ and $v \notin c\}$
        $L \leftarrow L - \{v, comp(v)\}$
    If $I = \phi$ then return("satisfiable")
    Else return("give up")
```

A_1 is very fast as it never assigns more than one value to each variable in I. Implicit in A_1 is the assignment of value *true* to literal v and the assignment of *false* to $comp(v)$. Therefore A_1 implicitly finds a solution if it does not "give up".

Recently we have obtained the result that algorithm A_1 efficiently solves SAT (does not "give up") in probability under the random clause model when $p = c\ln(n)/r$, $.5 < c < 1$, and $\lim_{n,r\to\infty} n^{1-c}/r \to 0$ [16]. In [16] it is also shown that instances generated according to the random clause model have no solution with probability tending to 1 when $p = c\ln(n)/r$, $.5 < c < 1$, and $\lim_{n,r\to\infty} n^{1-c}/r \to \infty$. Note that A_1 has good probabilistic performance even when variables appear in $O(n\ln(n)/r)$ clauses on the average.

These results are significant for two reasons. First, they say that A_1 almost always finds a solution to a random instance of SAT (generated under the random clause model) when one exists. Second, they demonstrate the power of the following line in A_1:

If there is a single-literal clause $\{u\} \in I$ then $v \leftarrow u$

We have shown that A_1 performs very poorly (almost always gives up) without this line when $\ln(n)/(2r) < p < \ln(n)/r$

Other favorable results have been obtained under a fixed clause length distribution. The *fixed clause length model* is the distribution on instances of SAT where each instance has n independently selected *fixed length random clauses* of k literals. The probability that a random instance of SAT under the fixed clause length model is satisfiable tends to 0 as n and r tend to infinity if $n/r > -\ln(2)/\ln(1-2^{-k})$. Furthermore, the average number of solutions to a random instance of SAT under this model is exponential in r if $n/r < -\ln(2)/\ln(1-2^{-k})$. The probability that a random instance of SAT under the fixed clause length model has at least one solution tends to 1 if $n/r < -f(k)/\ln(1-2^{-k})$ where $f(k)$ is monotonically increasing with k toward an asymptotic value no greater than $\ln(2)$. Because the character of instances changes so abruptly here, we refer to the point $n/r = -f(k)/\ln(1-2^{-k})$ as the *flip point*. In these studies k is assumed to be independent of n and r. Thus it appears that the case where $\lim_{n,r\to\infty} n/r = \alpha$, where α is any constant greater than zero, is particularly important when considering the fixed length clause model.

A number of algorithms have been analyzed under the fixed clause length model. It can be shown that a randomly guessed truth assignment will almost never be a solution to a random instance of SAT under the fixed length clause model if $\lim_{n,r\to\infty} n/r = \alpha$ where $\alpha > 0$. However, according to results in [11], an algorithm based on the pure literal rule (a component of the well known Davis-Putnam procedure [8]) efficiently solves SAT in probability under the fixed length clause model when $\lim_{n,r\to\infty} n/r \leq 1$. Recently we have shown that this algorithm can solve SAT efficiently in probability under the fixed clause length model only if the limiting ratio n/r obeys $n/r < 1/(1-ke^{-kn/2r})$. This bound is close to 1 for even moderately large k; for example, if $k = 6$ then $n/r < 1.07$. These results are even more interesting in light of the observation that stripping each clause of all its literals except for two results in an instance of 2-SAT which can be solved in polynomial time [12] and which almost always has a solution when $n/r < 1$. Since a solution to such an instance of 2-SAT is also a solution to the stripped instance of SAT from which it was created and since almost all instances of 2-SAT have solutions when $n/r < 1$, the trivial method of stripping literals performs about as well under the fixed length clause model as the algorithm based on the pure literal rule. The trivial method of stripping literals and the algorithm based on the pure literal rule are both superior to an algorithm of [4] which partitions clauses of a given instance I into groups so that no two clauses of different groups share literals associated with the same variable, solves SAT for each group and combines the solutions to each group to get the solution to I

But, there are a number of algorithms that have been shown to perform much better probabilistically under the fixed length clause model. In [5] we showed that A_1 efficiently solves SAT in bounded probability under the fixed length clause model when

$$\lim_{n,r\to\infty} n/r < \frac{2^{k-1}}{k}\left(\frac{k-1}{k-2}\right)^{k-2}$$

Notice that the expression on the right side of the inequality is $-O(1/k)/\ln(1-2^{-k})$ if k is large.

This result is significant for two reasons. First, we cannot make the claim that A_1 almost always finds a solution to a random instance of SAT when one exists, as we could in the case of the random clause model, since there is a large gap between the flip point ($n/r = -O(1)/\ln(1-2^{-k})$) and the point where A_1 begins to work well probabilistically ($n/r = -O(1/k)/\ln(1-2^{-k})$) due to the $1/k$ factor which appears in the latter term. Furthermore, for that range of n/r over which A_1 is probabilistically efficient, it is only able to find solutions efficiently with bounded probability whereas A_1 finds solutions efficiently in probability (almost all instances) under the random clause length model. Thus, we see that, in some sense, the fixed clause length model generates harder instances than the random clause length model (at least as far as A_1 is concerned) and the results based on the latter distribution do not map precisely to the same kind of results based on the former.

We also studied the following generalization of A_1:

```
$A_2(I)$ :
    Repeat
        Let $c$ be a smallest clause in $I$
        Choose $u$ randomly from $c$
        Remove from $I$ all clauses containing $u$
        Remove from $I$ all occurrences of $comp(u)$
    Until $I$ is empty or there exist two complementary unit clauses in $I$
    If $I$ is empty Then return ("satisfiable")
    Otherwise return ("give up")
```

In [5] we showed that A_2 efficiently solves SAT in bounded probability under the fixed length clause model when

$$\lim_{n,r\to\infty} n/r < \frac{1.54 * 2^{k-1}}{k+1}\left(\frac{k-1}{k-2}\right)^{k-2} \qquad \text{for } 4 \leq k \leq 40$$

and efficiently solves SAT in probability under the fixed length clause model when

$$\lim_{n,r\to\infty} n/r < \frac{0.92 * 2^{k-1}}{k}\left(\frac{k-1}{k-2}\right)^{k-2} \qquad \text{for } 4 \leq k \leq 40.$$

These results are significant for three reasons. First, A_2 efficiently solves SAT in probability (almost always) over about the same range of n/r that A_1 efficiently solves SAT in bounded probability. Second, the range of n/r over which A_2 is probabilistically efficient is only slightly greater than the range of n/r over which A_1 is probabilistically efficient. Thus, although A_2 performs much better than A_1 probabilistically, there is still a wide gap between the flip point and the point at which A_2 begins to perform well. Third, and most important, A_2 and A_1 are vastly superior in probabilistic performance compared to algorithms that rely on certain greedy heuristics to select the next variable to assign a value to. An example of a greedy heuristic is "select the variable v for which the difference between the number of occurrences of the literal v and the literal v' in I is greatest and assign variable v the value which satisfies most clauses". However, as we will see below, greedy heuristics added to A_1 and A_2 improve the performance of those algorithms significantly, especially for the case $k = 3$.

In the case $k = 3$ (CNF expressions with three literals per clause are instances of the 3-Satisfiability problem which is also NP-complete) we have shown in [6] that the maximum occurring literal selection heuristic (if there are no single-literal clauses in I, select a variable randomly and assign it the value which satifies most clauses) used with A_1 efficiently solves SAT in bounded probability under the fixed length clause model when $\lim_{n,r\to\infty} n/r < 2.9$. In the case $k = 3$, A_1 efficiently solves SAT in bounded probability when $\lim_{n,r\to\infty} n/r < 2.66$ This may be compared with the flip point ($n/r = 4$).

From our analysis in [5] and [6] we have devised the following new algorithm for SAT:

```
$A_3(I)$ :
    Repeat
        If there is a single-literal clause $\{l\}$ in $I$ Then $u \leftarrow l$
        Otherwise $u \leftarrow l^*$ such that $l^* \in L$ and for all $l \in L$ $w(l^*) \geq w(l)$
        Remove from $I$ all clauses containing $u$
        Remove from $I$ all occurrences of $comp(u)$
        $L \leftarrow L - \{u, comp(u)\}$
    Until $I$ is empty or there exist two complementary Unit Clauses in $I$
    If $I$ is empty Then return ("satisfiable")
    Otherwise return ("give up")
```

where $w(l)$, the weight of literal l, is determined as follows:

Let c be a clause in I and let $\mu_j(c)$ be a weighting function mapping clauses to integers. Let us say that $\mu_j(c)$ is the weight of clause c at the end of the j^{th} iteration of $A_3(I)$. Initially $\mu_0(c) = 1$ for every clause $c \in I$. The clause weighting function is updated as follows: if l is the literal chosen on the j^{th} iteration, $W_j(l)$ is the total weight of clauses containing l at the start of the j^{th} iteration (these clauses will be removed) and $N_j(l)$ is the number of clauses containing $comp(l)$ at the start of the j^{th} iteration (one literal will be removed from each of these clauses) then $\mu_j(c) = \mu_{j-1}(c) + W_j(l)/N_j(l)$ if c contains $comp(l)$, $\mu_j(c) = 0$ if c contains l and $\mu_j(c) = \mu_{j-1}(c)$ otherwise. The literal weighting function is

$$w(l) = W_j(l)/N_j(l)$$

According to our experiments, A_3 solves SAT efficiently in bounded probability under the fixed length clause model when $\lim_{n,r\to\infty} n/r < 4$.

The significance of this result is that A_3 appears to efficiently solve almost all instances of 3-SAT. We hope to prove this result analytically and devise an extension to A_3 which will provide similar performance for any fixed value of k.

4. Verifying Unsatisfiability

Although much of our work has been directed toward finding algorithms that obtain solutions when they exist, we are also interested in algorithms that, in probability, efficiently verify the unsatisfiability of instances of SAT that are unsatisfiable. The problem of verifying unsatisfiability seems to be harder than the problem of finding a solution when one exists since verification seems to require examination of many truth assignments to make sure that none is a satisfying assignment. Algorithm A_4 below represents a class of algorithms for verifying unsatisfiability.

```
$A_4(I)$ :
    If there is a clause in $I$ which has value *false* Then return "unsatisfiable"
    If all clauses in $I$ are *true* Then return "satisfiable"
    Select an unassigned variable $v$ from $V$
    Let $I_1$ be the result on $I$ of assigning *true* to $v$
    Let $I_2$ be the result on $I$ of assigning *false* to $v$
    If $A_4(I_1)$ and $A_4(I_2)$ return "unsatisfiable" Then return "unsatisfiable"
    Else return "satisfiable"
```

The line "Select an unassigned..." allows A_4 to be any one of a wide class of algorithms for verifying unsatisfiability by allowing any variable selection heuristic. Furthermore, important algorithms for verifying unsatisfiability such as the Davis-Putnam procedure have the form of A_4.

Unfortunately, we have found that, regardless of the variable selection heuristic used, A_4 requires exponential time, almost always, to verify unsatisfiability if instances are generated according to the fixed clause length model with the ratio of n to r fixed (recall that this is the most important relationship between n and r). Although pessimistic, this result is important because it shows us where not to look for algorithms that verify unsatisfiability efficiently in probability.

5. Hamiltonian Circuits

Given a graph $G = (V, E)$ with $|V| = n$, a hamiltonian circuit in G is an ordering $< v_1, v_2, ..., v_n >$ of vertices in V such that, for all $1 \leq i < j \leq n$, $v_i \neq v_j$ and $< v_i, v_{i+1} >$ is an edge in the edge set E and edge $< v_n, v_1 >$ is in E. A graph that has a hamiltonian circuit is called hamiltonian. If a graph G is hamiltonian, we see from the definition of hamiltonian circuit that there is a way in G to traverse a sequence of adjacent edges in such a way that every vertex in G is visited once and only once. The problem of finding a hamiltonian circuit in a graph (if one exists) is a very important NP-complete problem that has received much study. In [1] an efficient algorithm for the Hamiltonian Circuit problem was introduced and it was shown that this algorithm almost always finds a hamiltonian circuit in random graphs that are hamiltonian. This result has been regarded as evidence that the Hamiltonian Circuit problem is tractable in some probabilistic sense.

The result of [1] is based on the following graph distribution (for undirected graphs): a random undirected graph has n vertices and each pair of vertices is connected by an edge with probability p independently of any other edge connections. This distribution is analogous to the random clause distribution for SAT. According to a result of [1] there is an efficient algorithm which finds a hamiltonian circuit in a random undirected graph in probability if $p > (1 + \epsilon)\ln(n)/n$ where ϵ is a small constant. It had already been known that no hamiltonian circuit exists in a random graph with probability tending to 1 if $p < \ln(n)/n$. Thus, the algorithm of [1] finds a hamiltonian circuit in nearly all graphs that have one. Compare this result with our result that solutions to instances of SAT generated according to the random clause model can be found efficiently in probability when $p > \ln(n)/r$ (p, r and n are the parameters of the random clause model - not the random graph model), random instances of SAT have no solution with probability tending to 1 if $p < \ln(n)/(2r)$, and if $p = c\ln(n)/r$, $.5 < c < 1$, solutions to random instances of SAT are found efficiently in probability if $\lim_{n,r\rightarrow\infty} n^{1-c}/r \rightarrow 0$ and no solution exists if $\lim_{n,r\rightarrow\infty} n^{1-c}/r \rightarrow \infty$.

As previously mentioned we have found that algorithms which work well probabilistically under the random clause model do not necessarily work well under the fixed clause length model. The reason appears to be that the random clause model generates lots of "easy" instances. We illustrate this as follows: Let $I(n, r, p)$ (or I when the parameters are obvious) refer to a random instance of SAT generated according to the random clause model with parameters n, r and p. If p is proportional to $1/r$ the average number of literals per clause is constant. But, with probability tending to 1 there is a zero literal clause in $I(n, r, O(1/r))$. Zero literal clauses cause I to be unsatisfiable. As p is increased to, say, $O(\ln \ln(n)/r)$ the average number of literals per clause is now tending to infinity as r tends to infinity. A huge number of literals per clause makes the job of finding a truth assignment that satisfies lots of clauses much easier since more opportunites for doing so arise. But, as those opportunities increase I remains unsatisfiable because there is still a zero literal clause in I with probability tending to 1. If we increase p to a value close to $\ln(n)/(2r)$ the opportunities for developing a truth assignment satisfying lots of clauses become overwhelming but with probability tending to 1 there exists a zero literal clause which keeps I unsatisfiable. Finally, if p is increased beyond the value which forces a zero literal clause to be in I with probability tending to 1, there is nothing to prevent I from being satisfiable and the huge average number of literals per clause allows finding a satisfying truth assignment easily.

This phenomenon does not occur with the fixed clause length model for SAT. The reason is that there is no possibility of zero literal clauses so instances are satisfiable even when the number of literals per clause is constant. But, if the number of literals per clause is constant there are relatively few opportunities for developing a truth assignment which satisfies all clauses; thus, it is hard to do so (instances are "hard"). The analog of the fixed clause length model for SAT is the k-regular graph model for (undirected) graphs. (A k-regular graph is an undirected graph such that every vertex has degree k. The k-regular graph model assigns equal probability to every n vertex k-regular graph.)

From the above discussion it seems that the satisfiable instances generated under the random clause model are easier to solve than many satisfiable instances generated under the fixed clause length model. The result is that some algorithms which work well probabilistically under the random clause model do not work well under the fixed clause length model. We have verified this for some algorithms for SAT. Since the fixed clause length and random clause models for SAT are analogous to the k-regular and random graph models, respectively, we decided to check whether the sensitivity of probabilistic performance to instance distribution observed for SAT holds for the Hamiltonian Circuit problem. So far our results are as expected: the algorithm of [1] performs poorly on instances of k-regular graphs which have hamiltonian circuits. In fact, the algorithm of [1] almost never succeeds in finding a hamiltonian circuit when one exists. Thus, the question of whether the Hamiltonian Circuit problem is tractable in some probabilistic sense must be reopened and reexamined. We have begun to do this by experimenting with other algorithms for the Hamiltonian Circuit problem. One of these performs much better than the algorithm of [1] but does not find hamiltonian circuits often enough to get excited over. A complete report on this matter will be prepared after more work can be accomplished.

6. Summary of New Results

We have obtained the following results during the past year:

1. Algorithm A_1 efficiently solves SAT (does not "give up") in probability under the random clause model when $p = c\ln(n)/r$, $.5 < c < 1$, and $\lim_{n,r\to\infty} n^{1-c}/r = 0$ [16]. In [16] it is also shown that instances generated according to the random clause model have no solution with probability tending to 1 when $p = c\ln(n)/r$, $.5 < c < 1$, and $\lim_{n,r\to\infty} n^{1-c}/r = \infty$.

   These results are significant for two reasons. First, they say that A_1 almost always finds a solution to a random instance of SAT (generated under the random clause model) when one exists. Second, they demonstrate the power of the following line in A_1:

   If there is a single-literal clause $\{u\} \in I$ then $v \leftarrow u$

   We have shown that A_1 performs very poorly (almost always gives up) without this line when $\ln(n)/(2r) < p < \ln(n)/r$.

2. From our analysis in [5] and [6] we have devised a new algorithm for SAT which we called A_3 in section 3. According to our experiments, A_3 solves SAT efficiently in bounded probability under the fixed length clause model when $\lim_{n,r\to\infty} n/r < 4$.

   The significance of this result is that A_3 appears to efficiently solve almost all instances of 3-SAT that are satisfiable.

3. We have shown that any algorithm for verifying unsatisfiability of the kind represented by A_4 requires exponential time with probability tending to 1 under the fixed clause length model if the ratio of n to r is fixed.

   This result is significant because it applies to a wide class of algorithms.

4. We have found that the algorithm of [1] for finding Hamiltonian Circuits in random graphs performs poorly on hamiltonian k-regular graphs. We have also found that another algorithm for obtaining hamiltonian circuits performs much better than the algorithm of [1] on k-regular graphs but not well enough to find hamiltonian circuits even most of the time.

From these results we see that an NP-complete problem that had been regarded as tractable in the probabilistic sense may not be tractable after all. If not, the reason may be that the distribution chosen in [1] for analysis is faulty in that it allows too many "easy" instances to be generated. These results point out the need for further research aimed at being able to distinguish distributions that generate many "easy" instances from those that generate mostly "hard" instances.

7. Publications During the Last Year

a. Chao, M. T. and Franco, J., "Probabilistic analysis of two heuristics for the 3-Satisfiability problem," *SIAM J. Comput.* **15** (1986), pp. 1106-1118.

b. Chao, M. T. and Franco, J., "Probabilistic analysis of a generalization of the Unit Clause Literal Selection Heuristic for the k-Satisfiability problem," submitted to *SIAM J. Comput.*

c. E. Choukmane and J. Franco, "An approximation algorithm for the Maximum Independent Set problem," to appear in *Networks*.

d. Franco, J., "On the probabilistic performance of algorithms for the Satisfiability problem," *Information Processing Letters* **23** (1986), pp. 103-106

e. Franco, J., "Probabilistic analysis of a class of algorithms for verifying unsatisfiability," in preparation.

f. Franco, J. and Ho, Y., "Probabilistic analysis of a heuristic for the satisfiability problem," submitted to *Discrete Applied Math.*

References

1. Angluin, D. and Valiant, L., "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings," *Proc. 9th Annual ACM Symposium on Theory of Computing* (1977) pp. 30-41.

2. Berman, L. and Hartmanis, J., "On Isomorphisms and Density of NP and Other Complete Sets," *SIAM J. Comput.* **6** (1977)

3. Brown, C. and Purdom, P. W., "Average Time Analysis of Backtracking," *SIAM J. Comput.* **10** (1981), pp. 583-593.

4. Chao, M. T., "Probabilistic Analysis and Performance Measurement of Algorithms for the Satisfiability Problem," Ph.D. thesis, Case Western Reserve University, Cleveland, Ohio (1984).

5. Chao, M. T. and Franco, J., "Probabilistic Analysis of a Generalization of the Unit Clause Literal Selection Heuristic for the *k*-Satisfiability Problem," Technical Report No. 165, Department of Computer Science, Indiana University (1985).

6. Chao, M. T. and Franco, J., "Probabilistic Analysis of two Heuristics for the 3-Satisfiability Problem," *SIAM J. Comput.* **15** (1986), pp. 1106-1118.

7. Coffman, E. G., Frederickson, G. N. and Lueker, G. S., "Probabilistic Analysis of the LPT Processor Scheduling Heuristic," in *Deterministic and Stochastic Scheduling*, D. Reidel (1982), pp. 319-331

8. Davis, M. and Putnam, H., "A Computing Procedure for Quantification Theory," *J.ACM* **7** (1960), pp. 201-215.

9. Franco, J., "On the Probabilistic Performance of Algorithms for the Satisfiability Problem," *Information Processing Letters* **23** (1986), pp. 103-106.

10. Franco, J. and Paull, M., "Probabilistic Analysis of the Davis Putnam Procedure for Solving The Satisfiability Problem," *Discrete Applied Math.* **5** (1983), pp. 77-87.

11. Franco, J., "Probabilistic Analysis of the Pure Literal Heuristic for the Satisfiability Problem," *Annals of Operations Research* **1** (1984), pp. 273-289.

12. Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman (1979).

13. Gill, J., "Computational Complexity of Probabilistic Turing Machines," *SIAM J. Comput.* **6** (1977).

14. Goldberg, A., "Average Case Complexity of the Satisfiability Problem," *Proc. 4th Annual Workshop on Automated Deduction* (1979), pp. 1-6.

15. Goldberg, A., Purdom, P. W. and Brown, C. A., "Average Time Analysis of Simplified Davis-Putnam Procedures," *Information Processing Letters* **15** (1982), pp. 72-75.

16. Ho, Y. and Franco, J., "Probabilistic analysis of a heuristic for the Satisfiability Problem," submitted to *Discrete Applied Math.*

17. Karmarkar, N., "Probabilistic Analysis of Some Bin-Packing Problems," *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science* (1982), pp. 107-111.

18. Karp, R. M., "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane," *Math. Oper. Res.* **2** (1977), pp. 209-224.

19. Karp, R. M., "Probabilistic Analysis of Some Combinatorial Search Problems," In: J. F. Traub (ed.), *Algorithms and Complexity: New Directions and Recent Results*, Academic Press (1976).

20. Purdom, P. W., "Search Rearrangement Backtracking and Polynomial Average Time," *Artificial Intelligence* **21** (1983), pp. 117-133.

21. Purdom, P. W. and Brown, C. A., "The Pure Literal Rule and Polynomial Average Time," *SIAM J. Comput.* **14** (1985), pp.943-953.

22. Rabin, M. O., "Probabilistic Algorithms," In: J. F. Traub (ed.), *Algorithms and Complexity: New Directions and Recent Results*, Academic Press (1976).

23. Schwartz, J. T., "Fast Probabilistic Algorithms for Verification of Polynomial Identities," *J.ACM* **27** (1980).